人员分工

一、总体分工

陈邱华：前后端实时通信交互核心开发

刘杨健：后端 API 接口开发与交互协调

沈智恺：数据库交互与数据管理

刘链凯：后端基础类与业务逻辑核心开发

周兴煜：前端 UI 设计与界面开发

廖雨龙：前端交互逻辑与状态管理开发

二、具体任务

陈邱华：前后端实时通信交互核心开发

- **实时通信逻辑开发**：基于 SignalR 或 WebSocket 技术，在后端 `GameHub.cs` 中实现实时画笔信息、聊天消息的接收与广播逻辑，确保数据能及时准确地从后端推送至前端。例如，处理前端发送的画笔笔触数据，调用 `BroadcastStroke` 方法将其转发给房间内其他玩家；处理聊天消息的广播，实现实时聊天功能 。
- **前端通信接口对接**：在前端开发用于与后端实时通信的接口，如封装 `SocketService.js`，负责建立与后端的连接、发送实时画笔信息和聊天消息，并接收后端推送的数据，然后将其传递给相应的前端组件进行展示，如将接收到的画笔信息传递给 `DrawingBoard.vue` 组件进行绘制。
- **实时通信测试与优化**：针对实时通信功能进行专项测试，模拟高并发场景下的消息发送与接收，排查数据丢失、延迟等问题，并进行性能优化，确保游戏过程中实时交互的流畅性。

刘杨健：后端 API 接口开发与交互协调

- **后端 API 设计与实现**：设计并开发后端与前端交互的各类 API 接口，包括用户相关接口（如 `UserController.cs` 中的注册、登录接口）、游戏房间相关接口（如 `GameRoomController.cs` 中的创建房间、加入房间接口）、游戏流程相关接口（如 `GameController.cs` 中的开始游戏、结束游戏接口）。确保接口设计符合 RESTful 规范，提供清晰的接口文档，便于前端对接 。
- **前端请求处理与响应**：在后端控制器中编写逻辑，处理前端发送的各类请求，调用相应的服务（如 `UserService.cs`、`GameRoomService.cs`）进行业务处理，并将处理结果封装成合适的数据格式返回给前端。同时，处理请求过程中的异常情况，返回合理的错误信息。
- **前后端联调**：主动与前端开发人员协作，进行前后端联调工作。在联调过程中，及时定位和解决接口参数不匹配、数据格式不一致等问题，确保前后端数据交互的正确性和稳定性。

沈智恺：数据库交互与数据管理

- **数据库设计与搭建**：根据项目类图设计数据库表结构，包括用户表、游戏房间表、游戏表、词汇表等，定义表之间的关联关系，确保数据库设计满足业务需求。使用数据库迁移工具（如 Entity Framework Core 的 Migration）进行数据库的创建和版本管理。
- **后端数据访问层开发**：实现后端与数据库交互的仓储类（如 `GameRepository.cs`、`WordRepository.cs`）和服务类（如 `GameRoomService.cs`、`UserService.cs` 中涉及数据库操作的部分），编写

数据访问方法，如 `GetAllWords`、`CreateRoomAsync` 等，使用 Entity Framework Core 进行数据库的增删改查操作。
- **数据维护与优化**：负责数据库的日常维护工作，包括数据备份、恢复策略的制定与执行。对数据库性能进行监控和优化，通过索引优化、查询语句优化等方式提高数据访问效率。

刘链凯：后端基础类与业务逻辑核心开发

- **后端基础类开发**：根据类图实现后端核心基础类，如 `User.cs`、`Player.cs`、`GameRoom.cs`、`Game.cs` 等，定义类的属性和方法，确保类的设计符合业务逻辑和数据模型要求。同时，处理类之间的继承和关联关系，如 `Player` 类与 `User` 类的映射关系。
- **业务逻辑实现**：在后端服务类（如 `GameService.cs`、`WordManager.cs`）中实现核心业务逻辑，如游戏流程控制（开始游戏、结束游戏、计算得分）、词汇管理（获取随机词汇、添加词汇）等。确保业务逻辑的正确性和完整性，与其他后端模块进行良好的协作。
- **代码规范与质量把控**：制定后端代码的编写规范，确保团队成员代码风格统一。对自己编写的代码以及团队其他成员涉及基础类和业务逻辑的代码进行代码审查，保证代码质量，提高代码的可维护性和可扩展性。

周兴煜：前端 UI 设计与界面开发

- **UI 设计**：使用设计工具（如 Sketch、Figma）进行前端 UI 设计，包括登录界面、注册界面、游戏房间列表界面、游戏主界面等。设计过程中注重用户体验，确保界面布局合理、色彩搭配协调、操作流程便捷，输出高质量的设计稿。
- **前端组件开发**：根据 UI 设计稿，使用 Vue.js 或 React 等前端框架开发相应的界面组件，如 `LoginPage.vue`、`GamePage.vue`、`DrawingBoard.vue` 等。在组件开发过程中，实现组件的交互效果和动画效果，提高用户界面的美观性和交互性。
- **样式与响应式设计**：编写 CSS 样式代码，对前端组件进行样式美化，确保界面在不同设备（如电脑、平板、手机）上具有良好的显示效果。采用响应式设计原则，使界面能够自适应不同的屏幕尺寸，提供一致的用户体验。

廖雨龙：前端交互逻辑与状态管理开发

- **前端交互逻辑实现**：在前端组件中实现除 UI 展示外的交互逻辑，如登录表单的验证逻辑、游戏房间列表的刷新逻辑、游戏过程中的操作逻辑（如玩家猜词、切换游

戏状态）等。确保交互逻辑的正确性和流畅性，提升用户操作体验。

- **前端状态管理**：使用 Vuex 或 Redux 等状态管理工具，实现前端应用的状态管理。定义应用的全局状态，如用户登录状态、游戏房间信息、游戏进行状态等，并编写相应的 mutations 和 actions 来更新状态，确保各个组件之间状态的一致性和数据共享的正确性。
- **前端测试与 bug 修复**：对前端功能进行全面测试，包括功能测试、兼容性测试、性能测试等。及时发现并修复前端代码中的 bug，提高前端应用的稳定性和可靠性，配合其他成员进行前后端联调测试，确保整个系统的正常运行。
-

三、涉及到的代码文件与代码功能详细说明

• 陈邱华：前后端实时通信交互核心开发

- **前端**：
  - `frontend/src/services/socketService.js`：负责封装实时通信逻辑，使用 WebSocket 或 SignalR 建立与后端 `backend/Hubs/GameHub.cs` 的连接。实现发送实时画笔信息（从 `frontend/src/components/game/DrawingBoard.vue` 获取数据）和聊天信息（从 `frontend/src/components/gameRoom/Chat.vue` 获取数据）到后端，同时接收后端推送的信息并分发给相应的前端组件。
  - `frontend/src/components/game/DrawingBoard.vue`：实现玩家作画功能，包含绘制相关的逻辑和 UI。当玩家进行绘制操作时，将画笔信息（符合 `backend/Models/Stroke.cs` 定义的结构，如坐标、笔触大小、颜色等）通过 `socketService.js` 发送给后端。同时，接收并处理由 `socketService.js` 转发的其他玩家的画笔信息进行绘制显示。
  - `frontend/src/components/gameRoom/Chat.vue`：实现聊天功能的组件，具有聊天界面的 UI 和交互逻辑。通过 `socketService.js` 发送用户输入的聊天消息（符合 `backend/Models/ChatMessage.cs` 定义的结构）到后端，并接收由 `socketService.js` 传递的后端广播的聊天消息进行显示。
  - `frontend/src/router/index.js`：在处理实时通信相关的页面切换时，该文件中的路由逻辑可能会与实时通信状态进行交互，例如在进入游戏房间页面时，确保实时通信连接已建立等，陈邱华可能需要关注这部分与实时通信的协同逻辑。

- frontend/src/store/modules/game.js：实时通信获取的游戏相关状态信息（如游戏开始、结束等信号）可能会更新该模块中的游戏状态数据，陈邱华需要确保实时通信与状态管理之间的数据传递和更新逻辑正确。

- **后端**：

  - backend/Hubs/GameHub.cs：作为实时通信的核心组件，继承自 SignalR 的 Hub 类。接收前端通过 socketService.js 发送的实时画笔信息和聊天消息，利用 SignalR 的广播机制将这些信息推送给除发送者之外的其他连接的前端客户端，实现实时同步。同时，处理与实时通信相关的连接建立、断开等事件。

刘杨健：后端 API 接口开发与交互协调

- **后端**：

  - backend/Controllers/UserController.cs：处理前端关于用户相关的各类请求，如用户注册、登录、更新个人资料、获取总分数、设置用户角色和设置在线状态等。接收前端通过 frontend/src/services/apiService.js 发送的用户相关数据（符合 backend/Models/User.cs 定义的结构），调用 backend/Services/UserService.cs 中的业务逻辑方法进行处理，然后将处理结果以合适的格式返回给前端。
  - backend/Controllers/GameRoomController.cs：负责处理游戏房间相关的前端请求，包括创建游戏房间、加入房间、离开房间、获取房间列表、开始游戏、结束游戏、暂停游戏、恢复游戏等。接收前端通过 apiService.js 发送的相关数据（如创建房间时的 backend/Models/GameRoom.cs 数据结构），调用 backend/Services/GameRoomService.cs 进行业务处理，并将响应结果返回给前端。
  - backend/Controllers/GameController.cs：处理与游戏流程相关的前端请求，如开始游戏轮次、结束游戏轮次、获取游戏状态、计算得分等。接收前端通过 apiService.js 发送的请求数据，调用 backend/Services/GameService.cs 进行逻辑处理，再将处理后的结果返回给前端。
  - backend/Controllers/WordController.cs（假设存在，未在原结构中明确，但可能需要处理词汇相关接口）：处理前端对词汇的请求，如获

取随机词汇、添加词汇、更新词汇等。接收前端通过 apiService.js 发送的词汇相关数据（符合 backend/Models/Word.cs 定义的结构），调用 backend/Services/WordManager.cs 进行操作，并返回相应结果给前端。

- backend/Services/UserService.cs：实现用户相关的业务逻辑，为 UserController.cs 提供支持。例如，处理用户注册时的用户名唯一性检查、密码加密等操作；在用户登录时进行身份验证等。
  与 backend/Repositories/UserRepository.cs（假设存在，未在原结构明确，但可用于用户数据访问）交互获取或保存用户数据。
- backend/Services/GameRoomService.cs：处理游戏房间的业务逻辑，供 GameRoomController.cs 调用。包括创建房间时的初始化操作、加入房间时的权限检查和玩家添加、开始游戏时的状态更新等。
  与 backend/Repositories/GameRepository.cs 交互进行游戏房间数据的存储和读取。
- backend/Services/GameService.cs：实现游戏相关的业务逻辑，为 GameController.cs 服务。如计算游戏得分、管理游戏轮次、更新游戏状态等。
  与 backend/Repositories/GameRepository.cs 和 backend/Services/WordManager.cs（获取游戏用词汇）等协作完成游戏业务处理。
- backend/Services/WordManager.cs：提供词汇管理的业务逻辑，为 WordController.cs 和 GameService.cs 等提供支持。包括获取随机词汇、添加新词汇、更新词汇信息等操作，
  与 backend/Repositories/WordRepository.cs 进行数据交互。

- **前端**：

  - frontend/src/services/apiService.js：封装与后端进行 HTTP 通信的逻辑，向 backend/Controllers 中的各个控制器发送请求。根据不同的请求类型，组织和格式化请求数据（如用户信息、游戏房间信息等），并处理后端返回的响应数据，将其传递给相应的前端组件（如 frontend/src/views/LoginPage.vue、frontend/src/views/GamePage.vue 等）。

沈智恺：数据库交互与数据管理

- **后端**：

- backend/Models/DbContext.cs：定义数据库上下文，配置数据库连接信息（从 backend/appsettings.json 中获取数据库连接字符串），映射 backend/Models 下的各个实体类（如 User.cs、Player.cs、GameRoom.cs 等）与数据库表之间的关系，是后端与数据库进行交互的基础组件。
- backend/Repositories/GameRepository.cs：针对 backend/Models/GameRoom.cs 模型，实现对游戏房间数据的增删改查操作。例如，创建新的游戏房间记录、查询房间列表、更新房间状态和信息、删除房间记录等。
  与 backend/Services/GameRoomService.cs 紧密协作，为其提供数据访问支持。
- backend/Repositories/WordRepository.cs：依据 backend/Models/Word.cs 模型，实现对词汇数据的管理，包括获取随机词汇、添加新词汇、更新词汇属性、删除词汇等操作。
  为 backend/Services/WordManager.cs 和 backend/Services/GameService.cs（在游戏需要获取词汇时）提供数据获取和存储功能。
- backend/Repositories/UserRepository.cs（假设存在，原结构未明确但用户数据访问需要）：根据 backend/Models/User.cs 模型，实现对用户数据的数据库操作，如插入新用户、查询用户信息、更新用户资料、删除用户等。为 backend/Services/UserService.cs 提供数据访问接口。
- backend/Repositories/PlayerRepository.cs（假设存在，原结构未明确但玩家数据访问可能需要）：基于 backend/Models/Player.cs 模型，实现对玩家数据的数据库操作，如添加玩家记录、查询玩家信息、更新玩家状态等。
  为 backend/Services/GameRoomService.cs（管理房间内玩家信息）等提供数据支持。
- backend/Repositories/ChatMessageRepository.cs（假设存在，原结构未明确但聊天记录存储需要）：按照 backend/Models/ChatMessage.cs 模型，实现对聊天消息数据的存储和查询操作，用于保存游戏房间内的聊天记录，供后续查询和展示。
- backend/Repositories/GameHistoryRepository.cs（假设存在，原结构未明确但游戏历史记录存储需要）：根据 backend/Models/GameHistory.cs 模型，实现对游戏历史记录的存储和查询，记录游戏的相关信息（如参与玩家、得分、使用词汇等），方便后续统计和分析。
- backend/database/migrations 目录下的相关文件：负责数据库结构的变更管理，记录数据库表的创建、修改、删除等操作历史。沈智恺需

要维护这些迁移文件，确保数据库结构与应用程序的模型定义保持一致，在应用程序部署或升级时正确执行数据库迁移操作。

刘链凯：后端基础类与业务逻辑核心开发

- **后端**：
  - backend/Models/User.cs：定义用户实体类，包含用户的基本属性（如 id、username、passwordHash、totalScore、userRole、isOnline 等）和相关方法（如 RegisterAsync、LoginAsync、UpdateProfileAsync 等）。为用户相关的业务逻辑（在 backend/Services/UserService.cs 中实现）提供数据结构和基本操作定义。
  - backend/Models/Player.cs：继承或关联 backend/Models/User.cs，定义玩家在游戏中的特定属性（如 role、hasGuessed、isCorrect 等）和方法（如 GetRole、UpdateGuessStatus 等）。用于描述游戏中玩家的状态和行为，为游戏房间管理（在 backend/Services/GameRoomService.cs 中涉及）和游戏逻辑（在 backend/Services/GameService.cs 中）提供基础数据模型。
  - backend/Models/GameRoom.cs：描述游戏房间的实体类，包含房间的属性（如 id、name、status、gameMode、players、chatHistory、gameConfig、isPrivate、roomPassword 等）和方法（如 Create、Join、Leave、StartGame 等）。是游戏房间相关业务逻辑（在 backend/Services/GameRoomService.cs 中实现）的核心数据模型。
  - backend/Models/Game.cs：定义游戏的实体类，包含游戏的属性（如 id、currentRound、maxRounds、remainingTime、correctGuessCount、drawerScore、gameStatus 等）和方法（如 StartRound、EndRound、CalculateScore 等）。用于实现游戏流程的控制和状态管理（在 backend/Services/GameService.cs 中实现）。
  - backend/Models/Word.cs：词汇实体类，包含词汇的属性（如 id、content、category、difficulty 等）。为词汇管理（在 backend/Services/WordManager.cs 中实现）和游戏中词汇的使用（在 backend/Services/GameService.cs 中）提供数据结构。
  - backend/Models/ChatMessage.cs：定义聊天消息的实体类，包含消息的属性（如 content、timestamp、sender 等）。用于存储和传输游戏

房间内的聊天信息，在实时通信（backend/Hubs/GameHub.cs）和聊天功能（frontend/src/components/gameRoom/Chat.vue）中发挥作用。

- backend/Models/GameHistory.cs：记录游戏历史信息的实体类，包含参与游戏的玩家列表（players）、玩家得分（scores）、使用的词汇列表（usedWords）、游戏开始和结束时间（startTime、endTime）以及游戏统计信息（gameStats）等属性。为游戏历史记录的存储和查询（可能在相关的 Repository 中实现）提供数据模型。
- backend/Models/GameStats.cs：定义游戏统计信息的实体类，包含平均猜词时间（averageGuessTime）、猜对率（correctRate）、热门词汇（popularWords）等属性。用于对游戏数据进行统计和分析，可能在生成游戏历史记录或提供游戏统计报表时使用。
- backend/Models/GameConfig.cs：游戏配置实体类，包含每轮游戏时间限制（roundTimeLimit）、词汇难度分布（wordDifficultyDistribution）等属性。用于存储游戏的配置信息，在游戏开始或设置时使用，影响游戏的规则和流程。
- backend/Models/Stroke.cs：表示画笔笔触的实体类，包含笔触的属性（如 id、coordinates、brushSize、color 等）。在实时绘图功能中，用于前端和后端之间传输画笔信息（如在 frontend/src/components/game/DrawingBoard.vue 和 backend/Hubs/GameHub.cs 之间）。
- backend/Models/Point.cs：定义坐标点的实体类，包含 x 和 y 坐标属性。用于描述画笔笔触中的坐标信息（在 backend/Models/Stroke.cs 中使用）。

周兴煜：前端 UI 设计与界面开发

- **前端**：
  - frontend/index.html：项目的入口 HTML 文件，构建页面的基本骨架，加载 frontend/src/main.js 以启动前端应用程序。周兴煜需要确保该文件的结构合理，能够正确引入后续的 JavaScript 和 CSS 资源，为整个前端界面的展示奠定基础。
  - frontend/src/App.vue：作为根组件，包含整个应用的布局框架。周兴煜需要设计和实现其整体布局，引入其他子组件（如 frontend/src/components/layout/AppHeader.vue 和 frontend/src/components/layout/AppFooter.vue），处理组件之间的嵌套关系和全局样式的应用，确保整个应用界面的统一性和协调性。

- frontend/src/components/auth/UserLogin.vue：用户登录界面组件，负责设计和实现登录表单的 UI。包括输入框、密码框、登录按钮等元素的样式、布局和交互效果，以及与登录相关的提示信息的展示，提升用户登录的体验。
- frontend/src/components/auth/UserRegister.vue：用户注册界面组件，实现注册表单的 UI 设计。涵盖输入用户名、密码、确认密码等输入框的样式和布局，以及注册按钮、注册协议等元素的展示和交互，引导用户完成注册流程。
- frontend/src/components/game/DrawingBoard.vue：游戏画板组件的 UI 设计。包括画布的样式、大小、位置，以及画笔工具（如画笔颜色选择、画笔粗细调节等）的 UI 展示和交互设计，为玩家提供直观、便捷的作画界面。
- frontend/src/components/game/GuessWord.vue：猜词组件的 UI 设计。设计用于显示当前需要猜测的词汇的相关提示信息（如词汇的类别、首字母等）的界面，以及猜测输入框、提交按钮等元素的样式和布局，帮助玩家进行猜词操作。
- frontend/src/components/game/GameResult.vue：游戏结果展示组件的 UI 设计。负责设计展示游戏结束后的结果信息（如玩家得分、获胜者、游戏用时等）的界面，采用合适的样式和布局突出显示关键信息，让玩家清晰了解游戏结果。
- frontend/src/components/gameRoom/GameRoomList.vue：游戏房间列表组件的 UI 设计。设计展示可用游戏房间的信息（如房间名称、房间状态、玩家数量等）的界面，包括列表的样式、房间项的布局和交互效果（如点击进入房间等），方便用户浏览和选择房间。
- frontend/src/components/gameRoom/GameRoomCreate.vue：创建游戏房间组件的 UI 设计。实现创建房间表单的 UI，包括房间名称输入框、房间模式选择、隐私设置、密码输入框等元素的样式和布局，以及创建按钮的交互设计，引导用户创建游戏房间。
- frontend/src/components/gameRoom/GameRoomDetail.vue：游戏房间详情组件的 UI 设计。设计展示游戏房间详细信息（如房间内玩家列表、聊天记录、游戏规则等）的界面，采用合理的布局和样式呈现这些信息，增强用户对房间的了解。
- frontend/src/components/gameRoom/Chat.vue：聊天组件的 UI 设计。包括聊天输入框、发送按钮、聊天消息列表的样式和布局，以及消息时间显示、消息发送者标识等细节设计，实现流畅的聊天交互界面。
- frontend/src/components/layout/AppHeader.vue：应用头部组件的 UI 设计。设计包含应用名称、导航菜单（如登录、注册、主页等链接）、用户头像等元素的头部界面，提供统一的导航和品牌展示。

- frontend/src/components/layout/AppFooter.vue：应用底部组件的 UI 设计。实现包含版权信息、联系信息、社交媒体链接等元素的底部界面，为应用提供完整的页面结构。
- frontend/src/styles/global.css：全局样式文件，定义整个应用的通用样式。周兴煜需要在这里设置字体样式、颜色主题、边距和间距等全局样式规则，确保各个组件的样式风格一致，提升应用的视觉效果。

廖雨龙：前端交互逻辑与状态管理开发

- **前端**：
  - src/router/index.js：负责配置前端的路由，定义不同页面（如 src/views/LoginPage.vue、src/views/GamePage.vue 等）的路由规则，实现页面之间的导航功能。
  - src/router/guards.js：路由守卫文件，处理路由导航过程中的权限验证、登录状态检查等逻辑，确保用户只能访问有权限的页面。
  - src/store/index.js：Vuex 状态管理的入口文件，整合各个模块的状态（如 src/store/modules/user.js、src/store/modules/gameRoom.js、src/store/modules/game.js），提供全局的状态管理机制。
  - src/store/modules/user.js：用户相关的状态模块，管理用户的登录状态、用户信息等，
    与 src/components/auth/UserLogin.vue 和 src/components/auth/UserRegister.vue 等组件进行交互。
  - src/store/modules/gameRoom.js：游戏房间相关的状态模块，存储游戏房间列表、当前房间信息等状态，
    为 src/components/gameRoom/GameRoomList.vue 和 src/components/gameRoom/GameRoomDetail.vue 等组件提供数据支持。
  - src/store/modules/game.js：游戏相关的状态模块，管理游戏的当前状态（如是否开始、当前轮次等）、游戏结果等信息，
    与 src/components/game/Game.vue 和 src/components/game/GameResult.vue 等组件交互。
  - src/views/LoginPage.vue：登录页的逻辑处理，
    与 src/store/modules/user.js 配合，处理用户登录的业务逻辑，如验证用户名和密码、更新用户登录状态等。
  - src/views/GamePage.vue：游戏页的逻辑处理，
    与 src/store/modules/game.js 和 src/store/modules/gameRoom.j

s 交互，实现游戏流程的控制，如开始游戏、结束游戏等操作，并更新相应的状态。